REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	12/2/96	Final Technical Report -1/15/93-8/31/96

4. TITLE AND SUBTITLE
THE COMPLEXITY OF LEARNING IN NEURAL NETWORKS - THEORY AND APPLICATIONS

5. FUNDING NUMBERS
F49620-93-1-0120

6. AUTHOR(S)
Santosh S. Venkatesh

AFRL-SR-BL-TR-98-
0322

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Pennsylvania
Department of Electrical Engineering
200 South 33rd Street
Philadelphia, PA 19104

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of the Air Force
Air Force Office of Scientific Research
Bolling Air Force Base, DC 20332-6448

nm

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unlimited

12b. DISTRIBUTION CODE

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

13. ABSTRACT (Maximum 200 words)

Randomized algorithms are proposed and analyzed for learning binary weights for a neuron and links to the theory of random graphs are established. Optimal stopping phenomena in gradient descent learning algorithms are characterized and explained in terms of a time-varying effective machine complexity. The finite sample performance of the k-nearest neighbor algorithm for pattern recognition is rigorously characterized. Tradeoffs in learning from mixtures of labeled and unlabeled examples are determined.

19980421 150

DTIC QUALITY INSPECTED 8

14. SUBJECT TERMS
Neural networks, PAC learning, machine complexity, optimal stopping, binary integer programming, randomized algorithms, nearest neighbor algorithm, side information

15. NUMBER OF PAGES
22

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

THE COMPLEXITY OF LEARNING IN NEURAL NETWORKS—THEORY AND APPLICATIONS

PI: Santosh S. Venkatesh
Department of Electrical Engineering
University of Pennsylvania
Philadelphia, PA 19104
E-Mail: venkatesh@ee.upenn.edu

AWARD NUMBER: F49620-93-1-0120

FINAL TECHNICAL REPORT

December 2, 1996

1 Introduction

This technical report summarizes our findings obtained under the aegis of the AFOSR grant in a variety of areas related to computation and learning with particular reference to neural network settings. The main results obtained have been in problems related to: analyzing and characterizing random graph phenomena in algorithms learning binary weights for neurons [1]–[7]; optimal stopping and effective machine complexity in network learning [18]–[28]; nonparametric learning, with specific reference to the finite sample behavior of the k-nearest neighbor classifier [9], [12]–[15]; learning from mixtures of labeled and unlabeled examples [10] [11]; and issues in enumeration, computation, and learning [8] [12], [16], [17]. We summarize our main findings in each of these areas eschewing technical detail and formal proofs of the various assertions. These may be found in the various papers referenced above which are listed at the end of the report.

2 Learning Binary Weights and Random Graphs

See [1]-[7].

Consider the following classical problem in mathematical programming. Write $\mathbb{B} = \{-1, 1\}$ for simplicity, and let $\mathbb{B}^n = \{-1, 1\}^n$ denote the vertices of the n-cube. Let $\mathcal{U} = \{\mathbf{u}^\alpha, 1 \leq \alpha \leq m\}$ be an m-set of vertices (patterns, examples, or simply, points) in $\mathbb{B}^n$ and let $\mathcal{L} = \{l^\alpha, 1 \leq \alpha \leq m\}$ be a corresponding set of signs (classifications, labels, or colors), $l^\alpha \in \mathbb{B}$, which two-colors the m-set of vertices $\mathcal{U}$ into "positive" and "negative" examples. We are interested in the following question: Can we find a hyperplane separating the positive examples from the negative examples?

More formally, for $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$, let us denote the standard inner product in $\mathbb{R}^n$ by $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^n x_i y_i$. We seek to find a vector of weights $\mathbf{w} = (w_1, \ldots, w_n)$ such that, for each example $\mathbf{u}^\alpha = (u_1^\alpha, \ldots, u_n^\alpha) \in \mathcal{U}$,

$$\langle \mathbf{w}, \mathbf{u}^\alpha \rangle = \sum_{i=1}^n w_i u_i^\alpha \begin{array}{l} < \\ > \end{array} \begin{array}{l} 0 \quad \text{if } l^\alpha = -1, \\ 0 \quad \text{if } l^\alpha = +1. \end{array} \tag{1}$$

If there exists such a $\mathbf{w}$, then the linear subspace (hyperplane) orthogonal to $\mathbf{w}$ will separate the positive examples from the negative examples.

Before we proceed further, let us get rid of a minor notational nuisance by agreeing to reflect all negative example about the origin. In particular, if $l^\alpha = -1$, set $\mathbf{u}^\alpha \leftarrow -\mathbf{u}^\alpha$ and $l^\alpha \leftarrow +1$. An examination of the first inequality in (1) shows that this is perfectly legitimate as $\langle \mathbf{w}, \mathbf{u}^\alpha \rangle < 0$ iff $\langle \mathbf{w}, -\mathbf{u}^\alpha \rangle > 0$. Thus, without loss of generality, we can assume that the m-set of vertices $\mathcal{U}$ consists of only positive examples. Our programming problem then is to determine a weight vector $\mathbf{w}$ such that

$$\langle \mathbf{w}, \mathbf{u} \rangle > 0 \qquad (\mathbf{u} \in \mathcal{U}), \tag{2}$$

if any such $\mathbf{w}$ exists. The two formulations (1) and (2) are equivalent. In the sequel it will be found natural to set up probability distributions on labeled examples $(\mathbf{u}, l)$ for the framework (1); notational convenience will dictate, however, that we proceed with the analysis using the induced probability distributions for the equivalent framework (2) wherein all negative examples are reflected around the origin.

If $\mathbf{w}$ is allowed to range over Euclidean n-space $\mathbb{R}^n$, the corresponding decision problem is simple: the problem is just an instance of LINEAR PROGRAMMING and has a polynomial time solution, for instance, using interior point methods.[1] Note that in this case, the transformed problem (2) can be posed in an equivalent geometric formulation: Does the convex hull of the m-set of vertices $\mathcal{U}$ contain the origin? If the answer is affirmative then (2) has no solution; if the answer is negative then a solution $\mathbf{w}$ to (2) exists.

The problem becomes much harder if putative solutions $\mathbf{w}$ are allowed to range only over the vertices $\mathbb{B}^n$ of the cube. The corresponding decision problem in this case is an instance of BINARY INTEGER PROGRAMMING which is known to be NP-complete. Our focus will be on algorithmic approaches to this problem. Before we proceed, let us recast the problem as a learning problem in neural networks in a form which will allow of a substantive generalization.

2.1 The Binary Perceptron

A formal neuron in the model of McCulloch and Pitts is a linear threshold element which produces as output the sign of a linear form of its inputs. Viewed as a logic element, a neuron characterized by real *weights* $\mathbf{w} = (w_1, \ldots, w_n)$ maps Boolean[2] inputs $\mathbf{u} = (u_1, \ldots, u_n) \in \mathbb{B}^n$ into Boolean outputs $v \in \mathbb{B}$ in accordance with the threshold rule:

$$v = \operatorname{sgn}\left(\sum_{i=1}^{n} w_i u_i\right) = \operatorname{sgn}\langle \mathbf{w}, \mathbf{u} \rangle = \begin{cases} -1 & \text{if } \langle \mathbf{w}, \mathbf{u} \rangle < 0, \\ +1 & \text{if } \langle \mathbf{w}, \mathbf{u} \rangle \geq 0. \end{cases} \tag{3}$$

It is not hard to see that neurons form a universal basis for Boolean functions; for instance, a two-input NAND is computed by a two-input neuron whose weights are identically -1, thus:[3]

$$\operatorname{NAND}(u_1, u_2) = \operatorname{sgn}(-u_1 - u_2).$$

[1] Bibliographical notes and references to original work may be found in the concluding section.

[2] It is simpler in this framework to adopt the convention that truth values are represented by $+1$ (for true) and -1 (for false) instead of the usual 1 and 0.

[3] We've explicitly made use of the definition $\operatorname{sgn} 0 = 1$. If this expedient definition is too much for the gentle reader to swallow, she may replace the two-input neuron by a three-input neuron, two of whose inputs are literals, with the third input a constant -1. With the selection of all weights -1 again, it is simple to see that $\operatorname{NAND}(u_1, u_2) = \operatorname{sgn}(-u_1 - u_2 + 1)$. There are advantages to considering this more robust representation in the presence of noise.

In particular, for any Boolean function there exists a network of formal neurons which computes it.

The power of the neural network computational paradigm rests in part on the fact that the basic linear threshold computational element is substantially more powerful, viewed as a logic element, than standard logic elements. Indeed, a classical result asserts that of the 2^{2^n} possible Boolean functions of n Boolean variables, a formal neuron can compute $2^{\Theta(n^2)}$ Boolean functions by varying the weights. This investiture of substantial computational power in each element potentially results in significant improvements in circuit complexity.

The second leg of the appeal of networks of formal neurons (and sigmoidal generalizations where the sign operation is replaced by a sigmoidal function) as a universal computational paradigm rests on the availability of simple learning algorithms such as the Perceptron training algorithm (for a single neuron) and Backpropagation (for sigmoidal networks) which the circuit designer can utilize to prescribe the weights of the network.

While most uses of neural networks involve selections of real weights for the network, it is clear from the NAND example that binary weights suffice. Let us call the computational device represented by the input-output relation (3) a *binary neuron* or *binary perceptron* if the weights are constrained to take values in $\mathbb{B}$ only. It is then clear that a binary neuron forms a universal basis for Boolean functions. Thus, any network of neurons with real weights can be replaced by an equivalent, albeit larger, network of neurons with binary weights. While implementational cost considerations favor networks with binary weights, the practical utility of networks of binary neurons will devolve in significant measure upon whether there exist efficient algorithms for specifying binary weights for a network. The general learning problem here can be stated as follows: Given a labeled set of vertices and a network architecture,[4] can we learn binary weights for the network which yield a consistent classification of the set of examples? In particular, the particular instance of the BINARY INTEGER PROGRAMMING problem embodied in (2) is a special case of the general problem: can we learn binary weights for a (binary) perceptron which yields a consistent classification of a given set of labeled vertices?

While the worst-case intractability of the particular problem of learning binary weights for a neuron also implies the worst-case intractability of the general problem of learning binary weights for networks, nevertheless, as we see in the sequel, we can design (randomized) algorithms which very efficiently learn binary weights consistently for *almost all* instances of the problem. We focus here on three recent algorithmic approaches that we have proposed and analyzed to learning binary weights for perceptrons. While the results will be specialized to the BINARY INTEGER PROGRAMMING problem, the principles behind the algorithms can be readily extended to the general problem of learning binary weights for fixed architectures of binary neurons.

The analysis of the peccadilloes of individual algorithms waits upon the specification of the probability space in which the labeled examples reside, and two distinct scenarios emerge depending on the mechanism by which the labeled examples are specified. The probability spaces will be found to arise most naturally in the setting of the original problem (1) so let us refer back to (1) for the nonce to set up the two statistical scenarios of interest; notational expedience will dictate, however, that we subsequently work with the induced distributions for the equivalent Problem (2) obtained by reflecting negative examples about the origin.

We assume in both scenarios that $\{(\mathbf{u}^\alpha, l^\alpha) : 1 \le \alpha \le m\}$, *the set of labeled examples corresponding to Problem* (1), *is drawn by independent sampling from a given probability distribution on* $\mathbb{B}^n \times \mathbb{B}$. *Furthermore, we assume, again for both scenarios, that the marginal*

[4] A network architecture is specified by an interconnectivity graph on a set of neurons (which comprises the vertices of the graph) together with the identification of input neurons and the output neuron.

distribution of the examples $\mathbf{u}^\alpha$ *is uniform over the vertices* $\mathbb{B}^n$ *of the cube.* Equivalently, the pattern components $\{u_i^\alpha, 1 \le i \le n, 1 \le \alpha \le m\}$ form an independent, identically distributed sequence of symmetric Bernoulli random variables satisfying

$$\mathbb{P}\{u_i^\alpha = -1\} = \mathbb{P}\{u_i^\alpha = +1\} = 1/2.$$

It suffices now to specify the conditional distribution of the labels l^α given the examples $\mathbf{u}^\alpha$. Two cases arise naturally: in the first, the labels l^α are specified independently of the corresponding examples $\mathbf{u}^\alpha$ from an arbitrary probability distribution on $\mathbb{B}$; in the second, given the examples $\mathbf{u}^\alpha$, the labels l^α are specified in a completely deterministic fashion according to the labeling of the examples by a fixed target perceptron $\mathbf{w}^s \in \mathbb{B}^n$. We consider these two cases in turn.

Random Problems In our context, we say that a binary integer programming Problem (1) is *random* if examples and their corresponding labels are specified independently of one another. In particular, the set of labels $\mathcal{L} = \{l^\alpha\}$ forms an i.i.d. sequence of random variables, independent of the set of examples $\{\mathbf{u}^\alpha\}$, and with a common (not necessarily symmetric) Bernoulli distribution on $\mathbb{B}$.

While it is clear that for any m-set of points $\mathcal{U}$ there exists at least one labeling $\mathcal{L}$ which is linearly separable (in the sense of (1)), it is not difficult to see that the converse is also true for large enough m. Indeed, when $m > n$ there necessarily exists at least one labeling which cannot be linearly separated by a binary weight vector. A simple counting argument establishes this: there are 2^m possible labelings of an m-set of points, and only 2^n distinct binary weight vectors. It is clear further that the probability that any independently specified coloring of the set of patterns is linearly separable becomes arbitrarily small when m becomes suitably large compared to n, i.e., with high probability there will exist no solution vector for (1). Consequently, *any* binary learning algorithm will ultimately falter when the set of random examples becomes large enough. (Of course, some algorithms may fail somewhat earlier, even when solutions exist, because of a deficiency inherent in the algorithm itself.) We can hence pose the following question of interest: *What is the "largest" sample size* m *for which a given algorithm will, with high probability, yield a solution vector?* Suitably formalized as an asymptotic threshold notion this measure of the *capacity function* of an algorithm is, loosely speaking, the "largest" size $m = m_n$ of random binary integer programming problem that can be accommodated by the algorithm.

Linearly Separable Problems At the other end of the spectrum, labels for randomly drawn examples for Problem (1) are specified in a fully-determined, example-dependent fashion according to an underlying *target perceptron* (or *concept*). In particular, let $\mathbf{w}^s \in \mathbb{B}^n$ be some fixed (but unknown) vertex. For $\alpha = 1, \ldots, m$, set

$$l^\alpha = \begin{cases} -1 & \text{if } \langle \mathbf{w}^s, \mathbf{u}^\alpha \rangle < 0, \\ +1 & \text{if } \langle \mathbf{w}^s, \mathbf{u}^\alpha \rangle \ge 0. \end{cases} \tag{4}$$

Thus, for any sample size m we are always guaranteed the existence of at least one solution vector satisfying (1), namely $\mathbf{w}^s$, and an exhaustive search algorithm, for instance, will always yield a consistent hypothesis whatever the sample size m. The previous notion of capacity (or "largest size" of random problem that an algorithm can accommodate) hence does not have a particularly interesting counterpart in this linearly separable setting. Intuition suggests, however, that for a large enough sample size m, a solution vector satisfying the loading problem (1) will also be a good approximation to the underlying concept $\mathbf{w}^s$

labeling the examples. We can hence pose the following question of interest: *What is the "smallest" sample size m for which a given algorithm will, with high probability, identify the unknown target perceptron $\mathbf{w}^s$ as the unique solution vector satisfying* (1)? Suitably formalized as an asymptotic threshold notion, we call this "smallest" sample size $m = m_n$ the *sample complexity function* of the algorithm.

This is a problem familiar from learning theory. The output of the binary perceptron $\mathbf{w} \in \mathbb{B}^n$ (where we've tacitly adopted the notational convenience of identifying a perceptron by its weight vector) is the Majority function

$$h_{\mathbf{w}}(\mathbf{u}) = \operatorname{sgn}\langle \mathbf{w}, \mathbf{u} \rangle,$$

and the functions to be learned from examples (the hypotheses) are exactly the set of functions $\{ h_{\mathbf{w}} : \mathbf{w} \in \mathbb{B}^n \}$, i.e., Majority functions of a set of literals. We may identify each hypothesis function $h_{\mathbf{w}}$ simply by the corresponding vertex $\mathbf{w} \in \mathbb{B}^n$. In this setting, the goal of learning is to produce a hypothesis $\mathbf{w}$ (as a function of the set of examples) which approximates the underlying target concept $\mathbf{w}^s$ which labels the examples.

In the sequel, we will place somewhat more stringent requirements on the learning process: we will require *perfect learning* (or *perfect generalization*) wherein the target concept $\mathbf{w}^s$ labeling the examples is *exactly* identified (with high probability) by the algorithm. The "smallest" sample size m for which the algorithm can achieve this identification with high probability is the "complexity" of sample demanded by the algorithm to achieve perfect generalization.

2.2 Harmonic Update

Consider the following on-line situation. The teacher generates a sequence of m positive examples $\mathbf{u}^1, \ldots, \mathbf{u}^m$ and presents them in turn to the learner. Each example is presented only once, and in particular, the training sequence $\{\mathbf{u}[t]\}$ consists of the examples $\mathbf{u}^1, \ldots, \mathbf{u}^m$ in succession: $\mathbf{u}[t] = \mathbf{u}^t$, $1 \leq t \leq m$. The learning sequence $\{\mathbf{w}[t]\}$ of weight vector updates consequently terminates in m steps with $\mathbf{w}[1] = (w_1[1], \ldots, w_n[1]) \in \mathbb{B}^n$ an arbitrary initial weight vector and $\mathbf{w} \equiv \mathbf{w}[m+1]$ the final weight vector returned by the algorithm. The lack of memory poses an immediate problem here: at each epoch t, the learner has access only to her current estimate of the weight vector $\mathbf{w}[t] \in \mathbb{B}^n$, the current example $\mathbf{u}[t] = \mathbf{u}^t \in \mathbb{B}^n$, and the current epoch $t \in \{1, \ldots, m\}$, and as each example is seen only once, memory cannot be built up, as in other on-line scenarios, by repeated presentations of the examples.

Harmonic Update is an on-line, homogeneous algorithm which dynamically incorporates long-term memory of each example—single presentation notwithstanding—in the (binary) weights by using auxiliary randomization to retain, at each epoch t, and for each $i \in \{1, \ldots, n\}$, a maximal amount of information about *each* of the pattern components u_i^s, $1 \leq s \leq t$ in the corresponding weight $w_i[t]$. The weight updates are iteratively specified as follows:

Algorithm H (*Harmonic Update*). Given examples $\mathbf{u}[t] = \mathbf{u}^t = (u_1^t, \ldots, u_n^t) \in \mathbb{B}^n$ presented sequentially at epochs $t = 1, \ldots, m$, the algorithm recursively generates a sequence of binary weight vectors $\mathbf{w}[t+1] = (w_1[t+1], \ldots, w_n[t+1]) \in \mathbb{B}^n$ where $\mathbf{w}[t+1]$ is a random function of $\mathbf{w}[t]$ and $\mathbf{u}[t]$ only. After m epochs, the algorithm returns the final weight vector $\mathbf{w} \triangleq \mathbf{w}[m+1]$ as a putative vertex solution to the system of inequalities (2).

H1. [Initialize.] Set $\mathbf{w}[1] = (w_1[1], \ldots, w_n[1])$ to be an arbitrary vertex in $\mathbb{B}^n$. Set $t \leftarrow 1$.

H2. [New example.]. Obtain example $\mathbf{u}^t$.

H3. [Reinitialize component index.] Set $i \leftarrow 1$.

H4. [Update memory components.] If $w_i[t] = u_i^t$, set $w_i[t+1] = w_i[t]$; else if $w_i[t] = -u_i^t$, set

$$w_i[t+1] = \begin{cases} -w_i[t] & \text{with probability } 1/t, \\ +w_i[t] & \text{with probability } 1 - 1/t. \end{cases}$$

H5. [Iterate.] Set $i \leftarrow i+1$. If $i \leq n$, go back to step H4; otherwise set $t \leftarrow t+1$. If $t \leq m$ go back to step H2; otherwise set $\mathbf{w} = \mathbf{w}[m+1]$ and terminate the algorithm.

The intuition behind this randomized procedure is to try to obtain as large a covariance between each weight w_i and each of the corresponding pattern components u_i^α as possible. If this is achieved, the summands in the sum $\sum_{i=1}^{n} w_i u_i^\alpha$ of (2) are more likely to be positive, so that the chances of the entire sum being positive are improved. Now, at any epoch t, weight $w_i[t]$ contains information about the ith components of the first $t-1$ patterns. If the ith component of example $\mathbf{u}^t$ has the same sign as $w_i[t]$ there is no problem. The difficulty arises if $w_i[t]$ and u_i^t have opposite signs: not changing the sign of $w_i[t]$ will then lose information about pattern $\mathbf{u}^t$, while changing the sign of $w_i[t]$ will result in a loss of information about the first $t-1$ patterns. The solution in this case is to change the sign of the weight probabilistically, and with increasing reluctance as time passes (when there is presumably considerable past history stored in the weight). The exact measure of this reluctance to change the sign of the weight is given probabilistically by the harmonic sequence $1/t$. The effect of this randomized procedure is to ensure that each weight retains an equal amount of information about the corresponding component of every pattern. In particular, we have uniformly large covariances $\mathbb{E}\, w_i u_i^\alpha = \Theta(m^{-1})$.

The following is a capsule summary of the main features of the algorithm. Formal proofs and technical details may be found in the papers listed at the end of the report.

- Harmonic Update algorithm is on-line, homogeneous, and randomized.
- The algorithm has particularly parsimonious space and time requirements: it requires a buffer memory of $2n + \lceil \log m \rceil$ bits and a time complexity of m epochs (or a serial time complexity of mn bit steps).
- Notwithstanding its low complexity of specification, the algorithm has a surprisingly large capacity $\sqrt{n}/\sqrt{\log n}$ for random problems.
- The algorithm abjectly fails to generalize, however, and, even in the linearly separable case, is inconsistent with probability one on the sample for sample sizes exceeding $\sqrt{n}/\sqrt{\log n}$; its complexity function for perfect generalization (for linearly separable problems) is hence infinite.

In lieu of formal proofs of these asymptotic (as $n \to \infty$) results, the reader may wish to examine the trends which are apparent in Figures 1 and 2. In both cases, the consistency probability of the algorithm on the sample is estimated by the relative frequency of the number of times a solution was obtained for the loading problem (2) in 1000 independent runs and is shown plotted against normalized sample size $\frac{m}{\sqrt{n}/\sqrt{\log n}}$ with dimensionality n as a parameter. Observe the emergence of a threshold phenomenon when m is near $\sqrt{n}/\sqrt{\log n}$ for large n both for the random and the linearly separable case. The formal proofs devolve upon a consideration of the extreme tails of multivariate random walks.

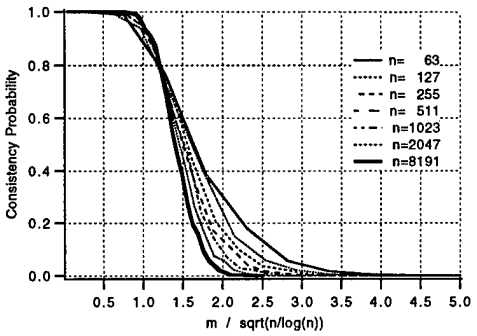

Figure 1: Computer simulations of Harmonic Update on random problems.

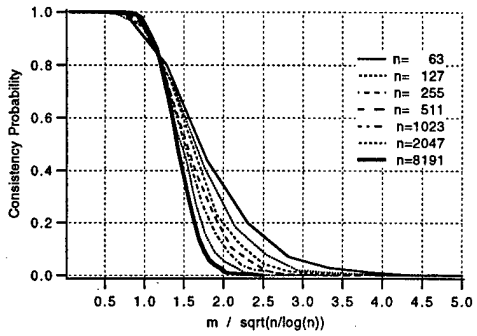

Figure 2: Computer simulations of Harmonic Update on linearly separable training sets.

2.3 Directed Drift

The failure of Harmonic Update to generalize efficiently from a linearly separable sample can be traced directly to its lack of consistency on large sample sizes; when m exceeds $\sqrt{n}/\sqrt{\log n}$, the algorithm does not build up sufficient memory about the sample in the weights. Two distinct approaches may be adopted to shore up this paucity of memory at the expense of the rather parsimonious space and time complexity requirements of Harmonic Update. In one approach, we may choose to augment algorithm memory by increasing the space complexity of the algorithm, by say, moving to an off-line procedure; this approach is considered in the following section. In an alternative approach, we may choose to retain the on-line setting (with its low attendant space complexity requirements) and bolster algorithm memory of the sample by repeated presentations of each example, thus increasing the time complexity requirements of the algorithm. This is the approach considered in this section.

Recall that in our discussion of Harmonic Update, each example was presented exactly once to the algorithm. The gentle reader may verify with a quick calculation (or simulation) that allowing repeated presentations of examples to the algorithm does not improve its generalization performance. The Directed Drift family of algorithms that we consider next, is not quite so efficient in a single pass of the data, but eventually, on repeated presentations of the examples gradually builds up a satisfactory representation of the underlying concept. The cost is time.

Let $\mathcal{U}$ be any set of positive examples, and let $\{\mathbf{u}[t]\}$ be any training sequence in which each of the patterns in $\mathcal{U}$ appears infinitely often. Let $\{\mathbf{w}[t]\}$ denote a binary learning sequence. For each epoch, t, we denote by $I[t]$ the subset of indices for which the corresponding components of $\mathbf{w}[t]$ and $\mathbf{u}[t]$ are opposite in sign:

$$I[t] \triangleq \{\, i : w_i[t] \neq u_i[t] \,\}.$$

In the simplest version of Directed Drift, no more than a single component of the weight vector is updated per epoch.

Algorithm D (*Directed Drift*). Given any sequence of positive examples $\{\, \mathbf{u}[t], t \geq 1 \,\}$ drawn from $\mathbb{B}^n$, the algorithm produces a sequence of hypotheses $\{\, \mathbf{w}[t] \in \mathbb{B}^n, t \geq 1 \,\}$ on-line.

D1. [Initialize.] Set $t \leftarrow 1$ and select an arbitrary initial hypothesis $\mathbf{w}[1] \in \mathbb{B}^n$.

D2. [Is the hypothesis consistent on the example?] Set $Y \leftarrow \langle \mathbf{w}[t], \mathbf{u}[t] \rangle$.

D3. [If it ain't broke, don't fix it.] If $Y \geq 0$, set $\mathbf{w}[t+1] \leftarrow \mathbf{w}[t]$ and go to Step D5.

D4. [Update hypothesis.] Else (if $Y < 0$) denote by $J[t] \triangleq \{ j : w_j[t] \neq u_j[t] \}$ the subset of indices for which the corresponding components of $\mathbf{w}[t]$ and $\mathbf{u}[t]$ are opposite in sign, pick a random index $j[t]$ from the uniform distribution on $J[t]$, and form the new hypothesis $\mathbf{w}[t+1]$ according to the following rule:

$$w_j[t+1] \leftarrow \begin{cases} +w_j[t] & \text{if } j \neq j[t], \\ -w_j[t] & \text{if } j = j[t]. \end{cases}$$

D5. [Increment time and iterate.] Set $t \leftarrow t+1$ and go back to Step D2.

The intuition behind the algorithm is as follows. If a binary solution vector $\mathbf{w}^s \in \mathbb{B}^n$ exists [i.e., the exemplar sequence $\{\mathbf{u}[t]\}$ is drawn from some underlying positive half-space $\mathbb{B}^n_+(\mathbf{w}^s)$] then necessarily we must have $\langle \mathbf{w}^s, \mathbf{u} \rangle = \sum_{i=1}^n w_i^s u_i \geq 0$ for each example $\mathbf{u} \in \mathcal{U}$. As there is a contribution of $+1$ to the sum if two corresponding components of $\mathbf{w}^s$ and $\mathbf{u}$ have the same sign, and -1 if the signs are mismatched, it follows that the binary solution vector has more component sign matches than mismatches with *each* example in $\mathcal{U}$.

Now the algorithm updates the current hypothesis $\mathbf{w}[t]$ if, and only if, the hypothesis is not consistent on the current example $\mathbf{u}[t]$ in which case a randomly chosen mismatched component $j[t] \in J[t]$ of the offending hypothesis is flipped (i.e., aligned with the sign of the corresponding component of the example). Since, by definition, the number $|J[t]|$ of mismatched components exceeds $n/2$, by the pigeonhole principle there is at least one member of $J[t]$ for which the corresponding components of $\mathbf{u}[t]$ and $\mathbf{w}^s$ have the same sign. It follows that there is a positive probability that the hypothesis update is in the direction of the target binary perceptron $\mathbf{w}^s$. Indeed, this suffices to show that Directed Drift, with probability one, converges in finite time to a hypothesis consistent on any subset of positive examples $\mathcal{U}$ whose members appear infinitely often in the sequence $\{\mathbf{u}[t]\}$.

We can speed up convergence by considering a batch version of Directed Drift which substitutes a democratic voting procedure for Step D4.

Algorithm B (*Boosting algorithm*). The algorithm receives as input a hypothesis $\mathbf{w}$ and a batch of m positive examples $\mathbf{u}^1, \ldots, \mathbf{u}^m$ which forms the electorate. The algorithm distributes an n-dimensional ballot to each voter (example) and returns the results of the election $\mathbf{b} = (b_1, \ldots, b_n)$ where, for each i, b_i takes only integer values from 0 to m and is the tally of votes on the desirability of changing the sign of the ith component of the hypothesis.

B1. [Distribute the ballot.] For each $s = 1, \ldots, m$, form the Boolean vector $(l_1^s, \ldots, l_n^s)$ where, for $i = 1, \ldots, n$,

$$l_i^s = \begin{cases} 1 & \text{if } u_i^s \neq w_i, \\ 0 & \text{if } u_i^s = w_i. \end{cases} \tag{5}$$

B2. [Hold an election.] For each $i = 1, \ldots, n$, tally the vote

$$b_i = \sum_{s=1}^{m} u_i^s.$$

Return the results of the election $\mathbf{b} = (b_1, \ldots, b_n)$.

If a batch of $m = m_b$ examples can be called by the algorithm at need, we may replace Step D4 by the following more efficient Step D4′ yielding a batch incarnation of the plain vanilla Directed Drift algorithm:

D4′. [Update hypothesis.] Else (if $Y < 0$) call $m - 1$ additional examples and pass the hypothesis $\mathbf{w}$ and the batch of m positive examples $\mathbf{u}^t, \ldots, \mathbf{u}^{t+m-1}$ to Algorithm B as parameters. From the results of the election, $\mathbf{b} = (b_1, \ldots, b_n)$, returned by Algorithm B, pick an arbitrary index j in the set $\{k : b_k \geq b_i, 1 \leq i \leq n\}$. Set $w_j \leftarrow -w_j$ and leave the other components of $\mathbf{w}$ unchanged. Set $t \leftarrow t + m - 1$.

The following is a capsule summary of the main features of the algorithm. Formal proofs and technical details may be found in the papers listed at the end of the report.

- If $\mathcal{U} \subseteq \mathbb{B}_+^n(\mathbf{w}^s)$ is any subset of vertices whose members appear infinitely often in a sample $\{\mathbf{u}[t]\}$ of positive examples of a binary perceptron $\mathbf{w}^s$, then, with probability one for all batch sizes, Directed Drift converges in finite time to a hypothesis which is consistent on $\mathcal{U}$. The proof is based on the equilibrium probability distribution of the states of the finite Markov chain which represents the system.
- For random problems, Directed Drift (for all batch sizes) has a maximal capacity function linear in n.
- For linearly separable problems, there exists an absolute positive constant $\alpha_c = 1.44797\ldots$ such that the sequence $\alpha_c n$ is an (upper upper bound for the) complexity function of Directed Drift (for all batch sizes).
- For linearly separable problems, the on-line version of Directed Drift, Algorithm D with a batch size of 1, has an exponential expected mistake bound bounded below by $\alpha e^{\beta n} + \mathcal{O}(n)$ where $\alpha = 1.771866547\ldots$ and $\beta = 0.139232271\ldots$ are absolute positive constants.
- The batch version of Directed Drift generalizes perfectly with a linear expected mistake bound $\mathcal{O}(n)$ for a quadratic-log batch sample complexity $m_b = \mathcal{O}(n \log n)$.

Again, in lieu of the formal proofs which can be found in the papers listed at the end of the report, the reader may wish to examine the suggestive trends in Figures 3 and 4. The figures

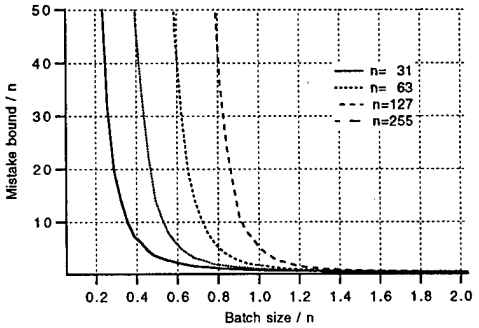

Figure 3: Computer simulations of the batch version of Directed Drift.

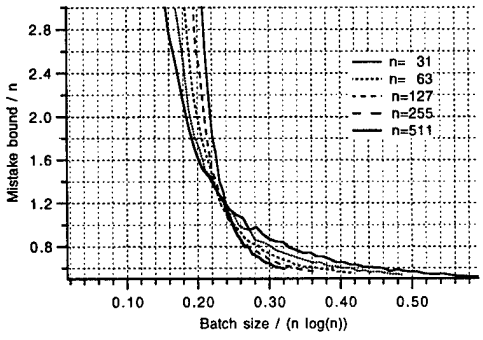

Figure 4: Numerical simulations suggest the necessary and sufficient batch size.

illustrate the savings in time that relatively small batch sizes can yield. For each n, average

mistake bounds until the algorithm pinpoints $\mathbf{w}^s$ are obtained from 1000 independent runs, and they are plotted against batch sizes. Both the mistake bounds and the batch sizes are normalized by the dimensionality n before plotting. For the values of n employed, all the (normalized) mistake bounds saturate below 1.0 for larger batch sizes, bespeaking a high probability of positive drift. On the other hand, as batch sizes decrease, the exponential convergence behavior of the algorithm becomes apparent, a consequence of the diminishing success probability. The question of interest is clearly to determine the tradeoff between batch size and the expected convergence time. In particular, how large must the batch size m_b be to ensure that a linear expected mistake bound is achieved? It transpires that a batch size $m_b = \mathcal{O}(n \log n)$ is sufficient to ensure a uniform positive drift probability of at least $\frac{1}{2} + \epsilon$ (and hence guarantee a linear expected mistake bound $\frac{n}{2\epsilon} + o(1)$). Numerical simulations suggest that a batch size of $\Theta(n \log n)$ is in fact both necessary and sufficient; see Figure 4. The new plot results from a slight transformation of Figure 3; batch sizes are normalized by $n \log n$ prior to plotting, and y-axis is zoomed in to show mistake bounds up to $3n$ only. Note that data curves of the five n values intersect near $(0.24, 1.2)$, where a threshold seems to emerge—if the batch size used exceeds $0.24n \log n$, the corresponding mistake bound appears to become arbitrarily close to $0.5n$ as $n \to \infty$; on the other hand, if the batch size employed falls below the critical value, the associated mistake bound seems to explode exponentially without bound as n increases. It follows that with a quadratic-log sample complexity $\mathcal{O}(n^2 \log n)$, batch version Directed Drift generalizes perfectly with a linear expected mistake bound $\mathcal{O}(n)$.

2.4 Majority Rule

If off-line procedures are permitted, substantial gains in capacity and learnability can be made as we illustrate in the following low-complexity off-line algorithm we dub Majority Rule. Let $\mathcal{U} = \{\mathbf{u}^1, \ldots, \mathbf{u}^m\}$ as usual denote the m-set of positive examples for loading problem (2). The algorithm prescribes weights as follows:

For $i = 1, \ldots, n$, *let*

$$U_i^+ = \{\mathbf{u} \in \mathcal{U} : u_i = +1\},$$
$$U_i^- = \{\mathbf{u} \in \mathcal{U} : u_i = -1\}.$$

Set

$$w_i = \begin{cases} +1 & \text{if } |U_i^+| \geq |U_i^-|, \\ -1 & \text{if } |U_i^+| < |U_i^-|. \end{cases}$$

In other words, $w_i = +1$ if patterns whose ith component is $+1$ are in the majority, and $w_i = -1$ otherwise. An examination of the Loading Problem (2) readily yields the intuitive basis of the algorithm. We require each of the sums $X^\alpha = \sum_{i=1}^n w_i u_i^\alpha$ to be positive and this is best brought about if each weight w_i is chosen so as to make it most likely that the summand $w_i u_i^\alpha$ is positive, i.e., if w_i and u_i^α are likely to agree in sign. Note that the algorithm is homogeneous and operates off-line.

The following is a capsule summary of the main features of the algorithm. Formal proofs and technical details may again be found in the papers listed at the end of the report.

- The memory requirement (space complexity) of Majority Rule is $mn+n$ bits (mn bits for the training set, n bits for the hypothesis weight vector). For each component of the binary neuron, the algorithm performs at most $m-1$ additions of bits and one sign operation. Thus the (serial) time complexity is mn, also the minimum possible. As in

the case of Harmonic Update, locality and homogeneity of Majority Rule allow us to speed up the hypothesis generation process by running n instances of the algorithm, one for each component, in parallel.

- The sequence $n/\pi \log n$ is a capacity function of the Majority Rule algorithm for random problems. Figure 5 provides empirical support of the asymptotic theoretical result. In the figure, the relative frequency of the number of times a solution was

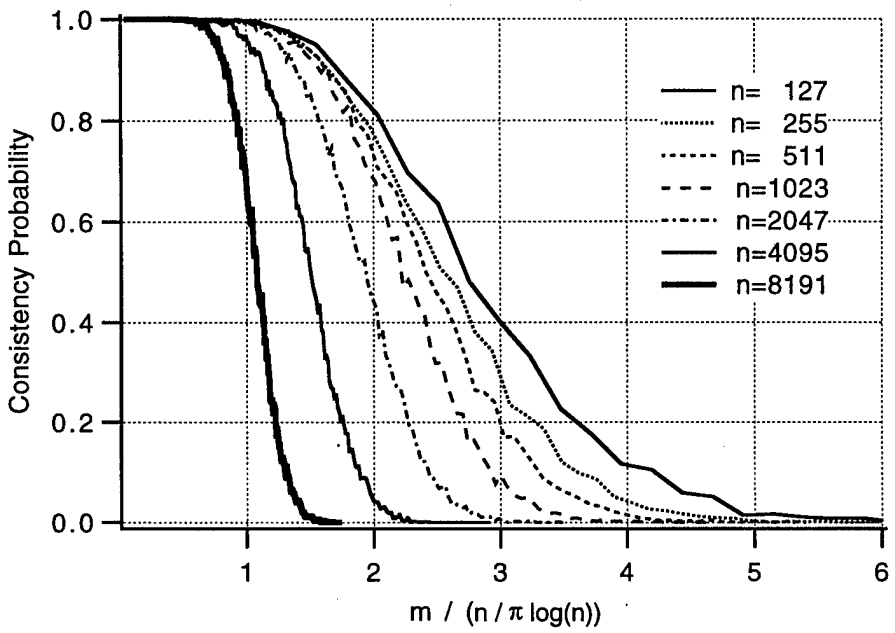

Figure 5: Computer simulations of Majority Rule on random problems. The consistency probability $P_{MR}(n, m)$ is plotted versus sample size m (normalized by $n/\pi \log n$) with n as a parameter.

obtained for the loading problem (2) in 1000 independent runs is shown plotted against normalized sample size $\frac{m}{n/\pi \log n}$ with dimensionality n as a parameter. Observe the emergence of a threshold phenomenon when m is near $n/\pi \log n$ for large n.

- For linearly separable problems, Majority Rule is *not* consistent on the examples. That is, if Majority Rule is applied to the m-sample set $\mathcal{U}$, the perceptron $\mathbf{w}$ generated does not necessarily satisfy (2). In other words, the algorithm does not always load the sample set.[5] Nevertheless, it exhibits superior generalization performance with low sample complexity. More specifically, we show that the probability of error incurred in estimating the target perceptron $\mathbf{w}^s$ by the hypothesis $\mathbf{w}$ can be made as small as

[5]While learning theory predicts that any consistent algorithm will generalize given a set of examples in excess of the VC-dimension, practical time constraints on the algorithms force many heuristics in use to be inconsistent. Examples of inconsistent, nonetheless popular, learning algorithms abound, and include such generic classes as gradient descent algorithms. For the present problem too, the NP-completeness of binary integer programming makes it unlikely that there is any polynomial time consistent algorithm which is guaranteed to work on all instances of the problem.

we choose with only a linear number of examples $m = cn$. Furthermore, the error probability decreases *exponentially* in m/n.

- As n approaches infinity, the Majority Rule algorithm exhibits an abrupt "phase transition" to *zero-error, perfect generalization* at when the number of examples (the *sample complexity*) reaches the critical number $m = \pi n \log n$. That is, it exactly learns the underlying majority function (i.e., the target perceptron $\mathbf{w}^s \in \mathbb{B}^n$) with high confidence when m exceeds $\pi n \log n$. On the other hand, when the number of examples is fewer than $\pi n \log n$, zero-error generalization is not achievable. Thus asymptotically, the algorithm needs sample complexities for perfect learning only slightly in excess (within a logarithmic factor) of the minimum needed to identify the function.

- Because the examples are drawn uniformly from $\mathbb{B}^n$, the perfect generalization result has the interesting implication that almost all instances of the binary integer programming problem are tractable when m exceeds $\pi n \log n$.

In Figures 6 and 7, compelling empirical evidence is presented in support of the strange peregrination of the algorithm towards perfect generalization. In both figures, an empirical estimate of the probability $P_{MR}(n, m)$ that the algorithm is consistent on a random, linearly separable m-sample is plotted versus (normalized) sample size ($\frac{m}{n/\pi \log n}$ in Figure 6 and $\frac{m}{\pi n \log n}$ in Figure 7). In brief, the algorithm is asymptotically consistent when the

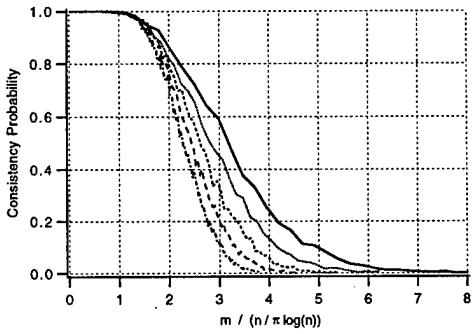

Figure 6: Simulations of Majority Rule on linearly separable problems: small sample regime.

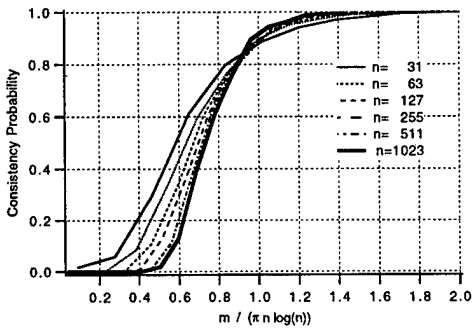

Figure 7: Simulations of Majority Rule on linearly separable problems: large sample regime.

sample size is less than the first threshold of $\frac{n}{\pi \log n}$, fails abruptly, i.e., is not consistent on the sample, when the sample size exceeds the first threshold, but recovers triumphantly from the dead when the sample size exceeds the second threshold of $\pi n \log n$. We dub this idiosyncratic behavior *asymptotic consistency*. At the second threshold—the sample complexity $\pi n \log n$ at which consistency secures so remarkably—exceeds the $1.45n$ information-theoretic estimate of Lyuu and Rivin (albeit only by a logarithmic amount) of the sample complexity at which the target perceptron is identified uniquely. Consequently, this implies that Majority Rule will in fact generate the target perceptron, i.e., generalize perfectly, when m exceeds $\pi n \log n$. The proofs of these results devolve upon a Poisson clumping argument in the very large deviation tails of multivariate random walks.

3 Optimal Stopping and Machine Complexity in Learning

See [18]-[28].

The primary goal of learning in a network setting is to find a network that gives valid generalization. In achieving this goal, the network designer is typically faced with the problem of appropriate network size selection so as to optimize the central trade-off between training error and network complexity. This issue has drawn much research effort in recent years and manifold principles, theories, and intuitions, including Occam's razor, and statistical model selection criteria such as Akaike's information criterion (AIC), Rissanen's minimum description length principle (MDL), and many others such as Barron's complexity regularization method have been developed. These principles all qualitatively support the following suggestive prescription from Vapnik-Červonenkis (VC) theory: between two machines which have the same empirical error, the one with smaller VC-dimension generalizes better. These methods, however, are typically valid only at the global minimum of the empirical error function (for example, the likelihood function for AIC), and furthermore, do not necessarily lead to optimal (or even nearly optimal) generalization performance. Two central issues remain improperly understood: (1) How is the generalization error affected by network complexity when there are a finite number of training examples? (2) How does the generalization performance evolve in time as training progresses? We have addressed these issues (and in some cases found comprehensive solutions) in the context of learning a target machine from a finite collection of random examples generated by it. We summarize our main results below.

Generalization performance is not overtly affected by network complexity (size) when the sample size is infinite. Indeed, at least nominally, in the limit of large sample sizes, valid generalization requires that learning be consistent, or at least approach the best approximation to the target function in the model class. However, network complexity manifests itself explicitly in the generalization performance when the number of available examples is *finite*. Given a fixed, finite sample, how then should one go about finding a network of proper complexity for which the generalization error is minimized? This is the crux of the first issue raised above. A satisfactory solution to this problem requires the explicit characterization of the effect of the learning algorithm on the "effective" complexity of the network.

The second issue relates to the time dynamics of the learning process. On the empirical front, it has often been observed that during network training by gradient descent there exists a critical region in time at which the trained network generalizes best, following which period the generalization error actually increases. (This latter phenomenon is ofttimes referred to as "over-training" or "fitting the noise.") Furthermore, in this critical region, the size of the network appears to play little rôle in the (best) generalization performance (at least, as long as the network is large enough to load the examples). These observations seem to contradict the learning-theoretic interpretation of Occam's razor! Are these numerical results just a fluke? If not, when should one stop learning, and how should one define the complexity of a network and go about adjusting it so that optimal generalization performance is achieved? Although relevant learning processes have been treated by numerous authors, the formal theoretical study of these problems has been abeyant.

We attack these issues in the framework of learning in a general class of machines which return a (variable) linear form of a (fixed) set of nonlinear transformations of points in an input space. The study of this simple machine class contains the key elements for the study of the generalization dynamics of general nonlinear learning machines without too much of the focus obscured by technical detail. The machines under consideration are also interesting

in their own right, since many practical problems can be modeled by this machine class and the important special case of a class of depth-two neural networks with a fixed nonlinear front end and a variable linear output element reduces to this model; these structures are also related to the Φ-machines considered by T. M. Cover. In [22], [23], and [24] we report extensions of the theory to general parametric nonlinear learning machines (such as general feedforward neural networks) where the basis functions are not fixed. It transpires that the flavor of the main results derived in [27], [28] carry over to the general nonlinear case though the generalization behavior for the nonlinear case turns out to be much richer than for the linear case, largely because of the increased complexity of the error function.

The learning-theoretic issues raised above boil down to a single question: For a given learning algorithm, how does the generalization error depend on the network parameters and the number of examples at each epoch of the learning process? In a sense, the main focus of [27], [28] is in providing a comprehensive answer to this question. In this paper we develop a rigorous characterization of the time-dynamics of generalization (for the class of machines considered) when a finite sample of examples is available and training is carried out by minimization of the empirical error via gradient descent; the paper is devoted to a development of the main result and to an exploration of its ramifications to network complexity, optimal stopping, network size selection, and regularization. In the following we briefly sketch the principal results.

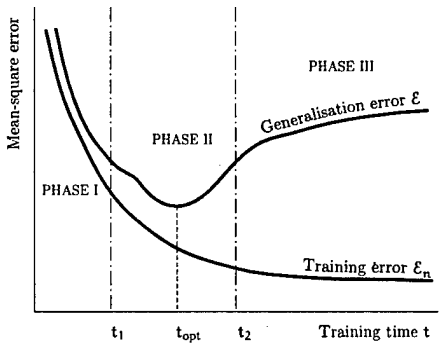

Figure 8: The three phases in the time evolution of generalization. (Not drawn to scale.) Best generalization obtains if training is optimally stopped during the second phase.

The empirical minimum mean-square estimate of the coefficient vector, which corresponds to the estimate obtained in the limit of training over an infinity of time steps, is unbiased and consistent. Should we then carry training out to its limit? Surprisingly, perhaps, the answer is "No." This assertion follows as a consequence of our MAIN THEOREM which relates the generalization error at each epoch of learning to that at the global minimum of the training error (in the limit of training). As a consequence of the theorem we are able to show analytically that as training progresses in time three distinct phases in generalization dynamics are evidenced. In the first phase, the generalization error decreases monotonically (keeping pace with a corresponding decrease in the training error); this phase is completed in $\mathcal{O}(\log n)$ time steps where n is the number of examples. The behavior grows more interesting in the second phase where the generalization error exhibits complex dynamics and an *optimal stopping time* t_{opt} is evidenced at which the smallest generalization error obtains; the second phase is also ephemeral and takes only $\mathcal{O}(\log n)$ time steps. Finally, in the third phase, the generalization error increases monotonically to a limiting value of error; this phase takes the rest of time. The three phases in generalization are indicated schematically in Figure 8. *Thus, best generalization occurs not at the limit of training when the global minimum of the training error is achieved, but rather after a finite number of steps of the order of the logarithm of the sample size.* Our main theorem also provides explicit bounds on the optimal stopping time and on the expected improvement in generalization error when training is optimally stopped.

One of the key concepts that emerges from our analysis is the formal notion of the *effec-*

tive size of a network. This is a time-varying, algorithm-dependent quantity which, in the limit of training over an infinity of time steps, coincides with the VC-dimension of the machine. Our main theorem allows us to characterize the relation between this time-dependent notion of machine size and generalization error. As is well known, a salient characteristic of neural networks is that they often involve a very large number of adjustable parameters as compared to traditional statistical models (such as classification and regression models) with a resulting large VC-dimension. For a given (small) sample of fixed size, how then does one explain empirical claims reporting valid generalization? Our results shed light on this puzzle: stopping learning at the optimal time results in a network with small complexity in the sense that its effective size at that time is typically substantially smaller than its effective size in the limit of training (the VC-dimension). *More generally, we show that the generalization error of the machine during the training process is determined at each training epoch by the effective size of the machine at that epoch rather than its VC-dimension.* Our analysis provides a formal framework within which optimal stopping can be viewed as dynamically fitting machine complexity to the sample wherein best generalization obtains when effective machine size best fits the sample size. Thus we rescue the prevailing intuition (Occam's razor) from its impending dilemma.

The study of generalization dynamics leads naturally to a practical *new network size selection criterion.* The new criterion allows us to simultaneously obtain estimates of optimal network size and optimal stopping time solely from a consideration of the given examples. We show that the new criterion has an optimal character and can be viewed as a generalization of Akaike's well-known information criterion to cover not just network complexity (the effective machine size in the limit of training) but the time evolution of the learning process as well. The analysis in [22] and [24] also allows of a comparison between various criteria and, in particular, leads to a better understanding of the relative merits of approaches based on the minimum description length principle vis à vis Akaike's information principle.

Regularization has been touted as a mechanism by which the performance of the gradient descent algorithm can be improved. Our analysis shows that the two methodologies of regularization and optimal stopping are distinct in spite of sharing superficial similarities. *In particular, for the general class of machines considered in [27], [28], we find that regularization frequently results in inconsistent learning with poorer generalization while optimal stopping always leads to consistent learning with improved generalization.*

Extensions of these results to general parametric nonlinear machines and neural networks and to other learning algorithms are detailed in [22], [23], and [24]. It transpires that the main results derived here carry over to the general nonlinear case even when the objective function is not restricted to the squared-error loss function. This is true for all results except those on the first phase of learning where different methods from those employed here must be used to study the problem. The generalization behavior for the nonlinear case turns out to be much richer than for the linear case, largely because of the increased complexity of the error function.

For practical applications of neural networks, our results demonstrate that training a network to its limit is not desirable. Thus, the inability of backpropagation networks to find the global minimum of the empirical error is not necessarily a drawback; it may precisely be the fact that global minima of the empirical error were not attained that has contributed to successful applications of neural networks in the past!

While the problem of inferring a rule from the observed data has been studied for a long time in learning theory as well as in other contexts such as in linear and nonlinear regression, the study of the problem as a dynamical process seems to open up a new avenue for looking at the problem. Many problems are still open, however, and we are currently attempting to extend our approaches to: systematically characterize and use side-information

in the learning problem; determine the appropriate node functions for a given problem; and characterize the effect of various nested architectures and learning algorithms.

4 Nonparametric Learning

See [9], [12]-[15].

A characteristic of nonparametric algorithms, such as the k-nearest neighbor classifier, kernel density estimation, and neural networks that acquire new hidden units as the training set is increased, is that they are able to learn solutions with nearly optimal accuracy using minimal prior information. Consequently, T. M. Cover has conjectured that the classification accuracy of the k-nearest neighbor algorithm estimates the accuracy of the best nonparametric classifier that uses the same set of labeled patterns. This conjecture rests on the proven asymptotic consistency of nearest neighbor algorithms, and the difficulty of incorporating side information into algorithms of such generality. That difficult pattern recognition problems usually do not exhibit useful side information, also suggests that the sample complexity of the k-nearest neighbor algorithm may estimate, in a loose sense, the intrinsic complexity of such problems. The k-nearest neighbor algorithm, in particular, is a close relative of unsupervised neural network learning algorithms such as Kohonen's learning vector quantization algorithm and feature maps. In [13] and [14] we provide perhaps the first rigorous estimates of the performance of the k-nearest neighbor algorithm on a *finite* sample of examples and an identification of optimal choice of metric. These results allow one to delineate in a fairly precise fashion how much information each example carries and yield qualitative information on the likely efficacy of growth algorithms in neural networks.

In the following we summarize the main results of [14]. In its original form, the k-nearest-neighbor classifier is perhaps the simplest pattern classification algorithms yet devised. Given a classification problem, in which a pattern, represented as an n-dimensional feature vector, is to be assigned to one of several pattern classes (e.g., states of nature), the k-nearest-neighbor algorithm requires three ingredients: (*i*) a finite reference sample of m correctly classified feature vectors from the given problem, (*ii*) an integer k (with $k \leq m$), and (*iii*) a metric, or pattern similarity function. Given an input feature vector, the k feature vectors from the reference sample that are closest to the input vector are identified using the metric. The input feature vector is then assigned to the class that appears most frequently amongst the k nearest neighbors. (If no single class appears with greatest frequency, then an auxiliary procedure can be invoked to handle ties.)

Despite its simplicity, this nonparametric algorithm is asymptotically consistent with a Bayes classifier, and, for a variety of practical applications, the accuracy of the algorithm is competitive with leading pattern classifiers. Its favorable performance in the infinite-sample limit ($m \to \infty$) was revealed in the seminal analysis of Cover and Hart in 1967. Their findings were later strengthened and generalized in a variety of studies which demonstrate that if the reference sample and input patterns are independently selected at random by arbitrary, stationary distributions; then the statistical risk of the k-nearest neighbor classifier (e.g., the probability that a random input vector is misclassified) tends to a value that is close to the Bayes risk (viz., the optimal misclassification probability). Specifically, as $m \to \infty$:

1. If $k = 1$, then the statistical risk tends to a value that does not exceed twice the Bayes risk; and

2. If $k \to \infty$ such that $k/m \to 0$, then the statistical risk tends to the Bayes risk.

In [14], we examine how the finite-sample risk of the k-nearest neighbor algorithm depends upon the selection of its metric. We specifically consider a weighted L_p metric of

the form $d(\mathbf{x},\mathbf{y}) = \|A(\mathbf{x}-\mathbf{y})\|_p$, where $\mathbf{x}$, $\mathbf{y} \in \mathbb{R}^n$, A is a constant, nonsingular, linear transformation, and $\|\mathbf{x}\|_p$ represents the L_p norm of $\mathbf{x}$. This classifier is then applied to a smooth family of classification problems $\mathcal{F}_N$ described by probability densities with uniformly bounded partial derivatives up through order $N+1$.

Under certain ancillary conditions on the class $\mathcal{F}_N$, we obtain an exact integral expression for the finite-sample probability of error of the k-nearest neighbor classifier by evaluating the error probability asymptotically using a multidimensional generalization of Laplace's method. This analysis yields a truncated asymptotic representation for the finite-sample probability of error in the form

$$P_m = P_\infty + \sum_{j=2}^{N} c_j\, m^{-j/n} + \mathcal{O}\left(m^{-(N+1)/n}\right). \tag{6}$$

Here, P_∞ is the infinite-sample risk probability of error derived by Cover and Hart (1967), and is independent of the choice of metric. The expansion coefficients $c_2, c_3, \ldots$, however, depend upon k, the metric parameters p and A, and the chosen procedure for handling ties, in addition to the probability distributions that describe the classification problem under consideration. This result applies to pattern recognition problems from $\mathcal{F}_N$ with a finite number of pattern classes and a deterministic tie-breaking algorithm. (Extensions to classifiers that use a random procedure (e.g., tossing a fair coin) to resolve ties can be obtained by averaging Eqn. (6) over an appropriately weighted ensemble of different deterministic procedures.) By analyzing the leading coefficients in (6), we establish the following:

> *For every C-class problem within $\mathcal{F}_2$, the weighted Euclidean metric (i.e., $p = 2$) minimizes the finite-sample probability of error, with respect to all other L_p metrics that use the same weight matrix A, if the size of the reference sample is sufficiently large.*

The asymptotic optimality of the Euclidean metric is shown in detail for a two-class problem with odd values of k. Under these conditions the evaluation of the expansion coefficients in (6) is simplified by the practical elimination of ties. For this case we have derived explicit formulas for the leading coefficients through c_5; coefficients for other cases can be obtained similarly. Numerical simulations of k-nearest neighbor classifiers also confirm that the L_2 metric is more accurate than the L_1 and L_∞ metric for normally distributed two-class problems in 12 and 25 dimensional feature spaces for practically attainable sample sizes.

The asymptotically optimal linear transformation A within the metric depends upon the underlying distributions that describe the given pattern recognition problem. Given the analytic forms of these distribution, this transformation can be obtained, in principle, as the solution of a constrained optimization problem using the Euler-Lagrange multiplier theorem.

To our knowledge, this is the first study to rigorously evaluate the finite-sample risk of the k-nearest neighbor classifier under different L_p metrics. In prior work, Fukunaga and Flick (1984) have nonrigorously examined the accuracy of this algorithm under a weighted Euclidean metric; their heuristic expression for the optimal quadratic weight matrix, however, differs somewhat from our estimates.

In ongoing work, we are seeking to further extend the range of validity of these results to general nonparametric situations, and in particular, learning algorithms in neural networks when the network structure is not held fixed, as is the case when lack of side-information about the problem constrains an effectively nonparametric search through network space.

These issues are also linked to those discussed earlier regarding net-size selection and optimum generalization for given data sets.

5 Learning from a Mixture of Labeled and Unlabeled Examples

See [10], [11].

The classical problem of *learning* a classification rule can be stated as follows: patterns from classes "1" and "2" (or "states of nature") appear with probabilities $p_1 = p$ and $p_2 = 1-p$, respectively; the pattern classes are represented by feature vectors x in a common N-dimensional Euclidean space $\mathbb{R}^N$, the patterns of class "i" distributed according to the class-conditional probability density $f_i(x)$ $(i = 1, 2)$. Labeled pairs $(x, y) \in \mathbb{R}^N \times \{1, 2\}$ are assumed generated according to the following mechanism: a pattern class (or "label") $y \in \{1, 2\}$ is first drawn randomly according to the distribution of classes $\{p_1, p_2\}$; a corresponding random feature vector $x \in \mathbb{R}^N$ is then drawn according to the class-conditional density f_y. In the supervised learning scenario, a *labeled* m-sample $\{(x_j, y_j), 1 \le j \le m\}$ is acquired by independent sampling from the distribution of pairs (x, y). Using the sample, the objective is to construct a decision rule which when presented with a random pattern x (drawn from the mixture density $f = p_1 f_1 + p_2 f_2$) produces a label which disagrees with the true class of origin by a probability P_{error} close to the minimal P_{Bayes} error rate. Formally this learning problem can be formulated in the framework of the Probably Approximately Correct (PAC) learning model of Valiant (1984) as follows: Given $\epsilon > 0$, $\delta > 0$, and a labeled sample of size $m = m(\epsilon, \delta)$, construct a classification rule which with confidence in excess of $1 - \delta$ has an error probability $P_{error} = P_{Bayes}(1 + c_0\epsilon)$.[6]

As is well known in the unsupervised learning literature an *unlabeled* teaching sample $\{x_i \in \mathbb{R}^N : 1 \le i \le n\}$ drawn by independent sampling from the mixture density $f(x)$, can also be used in the process of learning classification. If the mixture probability density function $f(x \mid \theta) = p_1 f_1(x \mid \theta_1) + p_2 f_2(x \mid \theta_2)$ is parametric (with unknown parameters $\theta = [\theta_1, \theta_2]$ and p) and is identifiable then the unlabeled sample $\{x_i\}$ can be used to generate an identifiable density function $f(x \mid \hat{\theta})$ as an estimate of $f(x \mid \theta)$. Identifiability ensures that the Bayes optimal decision border $\{x : p_1 f_1(x \mid \theta_1) = p_2 f_2(x \mid \theta_2)\}$ can be deduced if $f(x \mid \theta)$ is known and therefore one can construct an estimate of the Bayes border by using $f(x \mid \hat{\theta})$ instead of $f(x \mid \theta)$ (such plug-in rules are asymptotically optimal). Once the decision border is estimated, a *small* labeled sample suffices to learn (with high confidence and small error) the appropriate class labels $l_1, l_2 \in \{1, 2\}$ associated with the two disjoint decision regions in $\mathbb{R}^N$ generated by the estimate of the Bayes decision border.

Clearly, unlabeled examples, albeit of less worth than labeled examples, can still carry information of value to the learner. Furthermore, in many cases of practical interest, unlabeled examples exist in profusion in nature (or are inexpensively acquired) while their labeled counterparts are few in number (or are expensive to acquire). For example, in a two-class tree recognition problem "Douglas Fir" and "Spruce," unlabeled examples of these trees may exist in profusion in a forest but labeling them requires the services of a human expert who charges by the hour. Or, in an (idealized) cancer diagnosis scenario, obtaining x-rays of cells may be relatively cheap compared to the cost of the expert labeling them. A mixed sample learning system utilizing both labeled and unlabeled examples may hence be of interest where one trades off expensive labeled examples for (very many) unlabeled examples. In such a scenario, the learner would like to determine the exact tradeoff between

[6] We use $c_0, c_1, c_2, \ldots$ to represent positive constants independent of the error and confidence parameters ϵ and δ and the dimension of the feature space N.

labeled and unlabeled examples. This question was first posed by T. M. Cover in the following succinct but evocative form: *How many unlabeled examples is each labeled example worth*? In recent work Castelli and Cover have shown that if an infinity of unlabeled examples is available then for identifiable classes with p, f_1 and f_2 unknown, the probability of error decreases exponentially fast in the number of labeled examples to the Bayes risk; they have also shown that if only a *finite* number of labeled and unlabeled examples are available but the class-conditional densities f_1 and f_2 are known with the only unknown being the mixing parameter p, then under suitable regularity conditions labeled samples are exponentially more valuable than unlabeled samples.

In [10] we consider a parametric situation where the class-conditional densities $f_i(x \mid \theta_i)$ $(i = 1, 2)$ are specified by a parameter (vector) $\theta_i \in \mathbb{R}^N$. We assume that the learner is provided with a finite mixed sample of m labeled examples and n unlabeled examples together with side-information specifying the parametric *form* of the class-conditional densities (but not the densities themselves); the parameter vector $\theta = [\theta_1, \theta_2]$ as well as the mixing parameter p are, however, unknown to the learner. Our goal is to determine how the error rate depends on the sample sizes m and n and on the dimensionality N. Focusing on the Gaussian mixture case for definiteness, applications of uniform strong laws show that each labeled example has to be replaced by of the order of

$$c_1 \frac{N^2 \log \frac{1}{\epsilon}}{\epsilon^4 p^{11} \log N}$$

unlabeled examples to achieve the same performance. A precise statement of this result may be found in [10].

In nonparametric cases, the tradeoff between labeled and unlabeled examples is much more pronounced. For instance, if kernel density estimation is used to infer the mixture density of an identifiable family of bimodal mixture densities (including Gaussian mixtures), each labeled example is worth roughly

$$\frac{c_2 13^{N \log(5+\log N)}}{N \log N \epsilon^{\frac{N}{2 \log N}}}$$

unlabeled examples. A more precise statement of the result may be found in [11].

In ongoing work, we are attempting to build side-information about the problem into the learning framework.

6 Enumeration, Computation, Learning

See [8], [12], [16], [17

In tangential work, [8] provides a variety of results enumerating the number of folds in RNA secondary structures; [12] applies the probably approximately correct learning model of Valiant in conjunction with the uniform strong laws of Vapnik and Červonenkis to provide resolutions of the learnability and detectability of fraud in various situations; [16] provides an elementary proof using Pythagoras' theorem of the universal approximation capability of depth-two sigmoidal neural networks; and [17] provides a comprehensive general framework for storage capacity issues in recurrent neural networks.

7 Personnel

The following individuals were supported by and/or associated with various stages of the research effort: Sanjay Biswas, Selaka Bulumulla, Shao C. Fang, J. Stephen Judd, Vineet

V. Nene, Alon Orlitsky, Demetri Psaltis, Joel Ratsaby, Bharat Sarath, Robert R. Snapp, Santosh S. Venkatesh, and Changfeng Wang.

8 Technical Publications/Presentations

In the period January 16, 1993 through August 31, 1996 the following technical results obtained under the aegis of the AFOSR grant were disseminated to the scientific community in the form of technical papers submitted for publication to refereed journals, papers published in refereed conference proceedings, and also through the medium of invited papers and talks.

1. S. C. Fang and S. S. Venkatesh, "On the average tractability of binary integer programming and the curious transition perfect generalization in learning majority functions," in *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory*, Santa Cruz, California, 1993.

2. S. C. Fang and S. S. Venkatesh, "The capacity of Majority Rule," *Random Structures and Algorithms*, submitted February 1995, revised February 1996.

3. S. C. Fang and S. S. Venkatesh, "On batch learning in a binary weight setting," in *Proceedings of the IEEE International Symposium on Information Theory.* Whistler, Canada, September 1995.

4. * S. C. Fang and S. S. Venkatesh, "Learning finite binary sequences from half-space data," in *Proceedings of the International Conference on Neural Networks*, Perth, Australia, November 1995. [Invited talk.]

5. S. C. Fang and S. S. Venkatesh, "Learning binary perceptrons perfectly efficiently," *Journal of Computer and Systems Sciences*, vol. 52, 1996.

6. S. C. Fang and S. S. Venkatesh, "A threshold function for Harmonic Update," *SIAM Journal of Discrete Mathematics*, 1996.

7. S. C. Fang and S. S. Venkatesh, "Learning finite binary sequences from half-space data," *Random Structures and Algorithms*, submitted September 1996.

8. A. Orlitsky and S. S. Venkatesh, "On edge-colored interior planar graphs on a circle and the expected number of RNA secondary structures," *Discrete Applied Mathematics*, 1996.

9. D. Psaltis, R. Snapp, and S. S. Venkatesh, "On the finite sample performance of the nearest neighbor classifier," *IEEE Transactions on Information Theory*, 1994.

10. * J. Ratsaby and S. S. Venkatesh, "Learning from a mixture of labeled and unlabeled examples with parametric side-information," in *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, Santa Cruz, California, July 1995. [Long paper.]

11. * J. Ratsaby and S. S. Venkatesh, "The complexity of learning from a mixture of labelled and unlabelled examples," in *Proceedings of the Thirty-third Annual Allerton Conference on Communication, Control, and Computing*, Allerton, Illinois, October 1995. [Invited talk.]

12. B. Sarath and S. S. Venkatesh, "Auditor liability for management fraud," in *Proceedings of the Fifth Annual Conference on Intelligent Systems in Accounting, Finance, and Management*, Palo Alto, California, November 1993.

13. R. R. Snapp and S. S. Venkatesh, "Asymptotic predictions of the finite-sample risk of the k-nearest-neighbor classifier," *Proceedings of the 12th International Conference on Pattern Recognition*, vol. 2, pp. 1–7. Los Alamitos, California: IEEE Computer Society Press, 1994.

14. R. R. Snapp and S. S. Venkatesh, "k-nearest neighbors in search of a metric," in *Proceedings of the IEEE International Symposium on Information Theory.* Whistler, Canada, September 1995.

15. R. R. Snapp and S. S. Venkatesh, "k-nearest neighbors in search of a metric," *Annals of Statistics*, submitted July 1996.

16. S. S. Venkatesh, "On approximations of functions by depth-two neural networks," accepted for presentation at the *IEEE International Symposium on Information Theory*, Trondheim, Norway, 1994.

17. * S. S. Venkatesh, "Connectivity versus capacity in the Hebb rule," in *Advances in Neural Computing*, (eds. A. Orlitsky, V. Roychoudhuri, and K-Y. Siu). Norwell, Massachusetts: Kluwer Academic Publishers, 1994. [Invited contribution.]

18. * S. S. Venkatesh, "Learning dynamics," Colloquium at Dupont, Nemours, & Co., Inc., Wilmington, Delaware, April 1995. [Invited talk.]

19. * S. S. Venkatesh, "Finite sample effects in pattern recognition," *Workshop on Computational Learning*, Australian National University, Canberra, Australia, December 1995. [Invited talk.]

20. * S. S. Venkatesh, "Optimal stopping, effective machine complexity, and learning," Colloquium at Australian National University, Canberra, Australia, December 1995. [Invited talk.]

21. * S. S. Venkatesh, "Model selection and optimal stopping in learning," Electrical Engineering Colloquium, University of Queensland, Brisbane, Australia, December 1995. [Invited talk.]

22. C. Wang and S. S. Venkatesh, "Machine size selection for optimal generalization," *Workshop on Applications of Descriptional Complexity*, New Brunswick, New Jersey, July 1994.

23. C. Wang and S. S. Venkatesh, "Temporal dynamics of generalization in neural networks," in *Advances in Neural Information Processing Systems 7*, (eds. D. S. Touretzky, G. Tesauro, and T. K. Leen). Cambridge, MA: MIT Press, 1995.

24. C. Wang and S. S. Venkatesh, "Criteria for approximation error and complexity tradeoff in learning," in *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, Santa Cruz, California, July 1995.

25. C. Wang, S. S. Venkatesh, and J. S. Judd, "Optimal stopping and effective machine size in learning," in *Proceedings of the Sixth conference on Neural Information Processing Systems*, Denver, Colorado, 1993.

26. C. Wang, S. S. Venkatesh, and J. S. Judd, "Optimal stopping and effective machine complexity in learning," in *Advances in Neural Information Processing Systems 6* (eds. J. Cowan, G. Tesauro, and J. Alspector). San Mateo, California: Morgan Kauffman, 1994.

27. C. Wang, S. S. Venkatesh, and J. S. Judd, "Optimal stopping and effective machine complexity in learning," *IEEE Transactions on Information Theory*, submitted March 1995.

28. * C. Wang, S. S. Venkatesh, and J. S. Judd, "Optimal stopping and effective machine complexity in learning," in *Proceedings of the IEEE International Symposium on Information Theory.* Whistler, Canada, September 1995. [Long paper.]